EVALUATIONS

| | | After Similarity Evidence Only | | | | After Similarity and Covariation Evidence | | | |
| | | CAUSES | | | | CAUSES | | | |
COVARIATION	WORKER TRAINING/ LABOR VARIANCE SIMILARITY	Workload Schedule	Raw Material Quality	Worker Training	Manager Efforts	Workload Schedule	Raw Material Quality	Worker Training	Manager Efforts
Medium	High	-3.43 (5.65)	0.93 (4.38)	6.29 (0.99)	6.36 (1.32)	-3.36 (4.55)	1.07 (4.36)	6.57 (1.34)	6.43 (1.03)
Medium	Low	-3.29 (10.22)	6.64 (1.48)	1.14 (3.36)	5.71 (1.60)	-3.29 (10.22)	6.86 (1.21)	1.36 (3.02)	5.64 (1.32)
Weak	High	-3.21 (8.03)	0.86 (3.36)	6.71 (0.84)	6.07 (4.38)	-2.93 (6.53)	3.79 (1.80)	1.64 (9.32)	5.07 (8.99)
Weak	Low	-3.14 (10.29)	7.29 (1.76)	1.00 (6.15)	5.50 (1.50)	-2.86 (9.67)	1.93 (11.30)	3.93 (2.53)	5.64 (7.32)

Notes: Each cell contains 14 subjects. The boxed cells identify the cause/effect similarity manipulations.

BEBR

FACULTY WORKING
PAPER NO. 1082

On Finding Minimal Bottleneck Cells for
Grouping Component-Processor Families

Anthony Vannelli
K. Ravi Kumar

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign

BEBR

On Finding Minimal Bottleneck Cells for
Grouping Component-Processor Families

Anthony Vannelli
University of Toronto

K. Ravi Kumar, Assistant Professor
Department of Business Administration

On Finding Minimal Bottleneck Cells for Grouping

Component-Processor Families

Anthony Vannelli

Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

and

Department of Industrial Engineering
University of Toronto
Toronto, Ontario M5S 1A4

K. Ravi Kumar

Department of Business Administration
University of Illinois at Champaign-Urbana
Champaign, Illinois 61820

Abstract: The selection of component parts and processors, poses an important problem in the design and planning phases of cellular manufacturing and flexible manufacturing systems. In most real-life situations, this grouping invariably leads to "bottleneck" parts and processors. In this paper, we deal with the issue of identifying the minimal number of bottleneck cells (processors or parts) which, when dealt with either through duplication of processors or subcontracting of parts, will result in perfect component-processor groupings with no overlap. The polynomially bounded algorithms used in the analysis are oriented towards finding minimal cut-nodes in either partition of the bipartite graph.

1. Introduction

In this paper, we discuss the important problem of finding bottleneck machines that arises when grouping component-processor families. Ideally, in group technology (GT), one would like to find a dedicated cell of machines which can be grouped, tooled and scheduled as a single unit. Moreover, one would like to have the *least* interaction between such units. Our discussion focuses on efficiently implementing a grouping analysis using existing route sheet information. This approach was first proposed by (Burbidge, 1975) and has been investigated by several researchers (Basale and Tripathi, 1981; Kusiak, 1984; Chakravaty and Shtub, 1984; Kumar and Vannelli, 1984). The route sheet information allows one to model the component-processor grouping as a bipartite graph. If the bipartite graph has several natural groupings; i.e., disconnected subgraphs, then there are several efficient algorithms available for identifying the connected components of the given graph (e.g. Harary 1959, 1960; Ogbuobiri, *et al.*, 1970; Zaborsky, *et al.*, 1982; King 1979, 1982).

However, the majority of real-life component processor groupings (i.e. 200 machines and 2000 components) are connected to begin with or have a large grouping which is connected. In this case, one wishes to find the smallest number of exceptional components which if deleted would disconnect the graph. This problem is called the *tearing* problem, which was first investigated by Steward (1962, 1965). This problem can also be approximated by linear transportation problems (Barnes, 1982; Vannelli, 1984; Kumar and Vannelli, 1984).

A related problem is to find the least number of machines, which if duplicated or if scheduled properly would also disconnect the network. These machines are appropriately termed *bottleneck machines*. Equivalently, one can define the notion of *bottleneck components*. A bottleneck component is a component which normally uses a large number of machines to be completed. In this case, it may be more desirable to have such components completed outside the actual operation; that is, schedule

1

and process the other components first and leave the bottleneck components last or have bottleneck components produced by a supplier.

In this paper, we look at the problem of finding the minimal number of bottleneck machines and/or components while *disconnecting* the operation into several groups. We show that this problem is equivalent to finding the *minimal cutnodes* of a graph while disconnecting the graph into m subgraphs having at most k nodes each. We implement and extend a dynamic programming approach due to Lee, *et al.* (1979) to find bottleneck cells. This algorithm is a heuristic for finding minimal cutnodes of a graph given that the designer specifies the number of disconnected groups desired, an upper bound on the number of nodes in each group and a starting node in each group. The interactive capabilities of the Lee, *et al.* (1979) approach allows the design to consider a variety of *good* groupings. In the final analysis, a grouping may be chosen because of other desirable characteristics. The intent is to give the designer complete flexibility in the design and analysis of such systems. This is more in the spirit of currently evolving FMS technology.

This paper is divided into six sections. The bottleneck machine problem that arises in component-processor grouping is described in Section 2. A generalization to the bottleneck cell problem is introduced. A description of the minimal cutnode problem and its equivalence to the minimal bottleneck cell problem is given in Section 3. The heuristic used by Lee, *et al.* (1979) is outlined. A variation of this heuristic is developed for finding minimal bottleneck machines or equivalently, minimal bottleneck components is given in Section 4. Applications of the derived algorithm are presented in Section 5. Conclusions are drawn in Section 6.

2. Bottleneck Machine Problem in Component-Processor Grouping

In this section, we describe how bottleneck machine problems arise. This notion is generalized to the *bottleneck cell* problem, where a cell can be either machines, components, or both. The equivalence of the bottleneck cell problem and minimal cutnode problem is shown.

A major restriction in the efficient decomposition of the component-processor matrix into completely disconnected groups is that often a large number of components requires operations by very few machines. Another restriction is that a large number of machines is required to produce few components. King (1980) calls the former situation the *bottleneck machines* problem.

The 5 machine - 5 component example in Figure 1 illustrates the occurrence of bottleneck machines and components.

$$
\text{machines}\quad
\begin{array}{c}
1\\2\\3\\4\\5
\end{array}
\left[
\begin{array}{ccccc}
1 & 1 & & & 1\\
1 & & & & 1\\
 & & 1 & 1 & 1\\
 & & & 1 & 1\\
1 & 1 & 1 & 1 & 1
\end{array}
\right]
$$
$$
A\quad B\quad C\quad D\quad E
$$

components
Figure 1

Observe that if machine 5 did not have to operate on the 5 components and if component E did not have to be processed by the 5 machines, the network would become completely disconnected. By deleting row 5 and column 5, the machine-component matrix is disconnected into two mutually disconnected networks as in Figure 2.

$$
\text{machines}\quad
\begin{array}{c}
1\\2\\3\\4
\end{array}
\left[
\begin{array}{cc|cc}
1 & 1 & & \\
1 & & & \\ \hline
 & & 1 & 1\\
 & & & 1
\end{array}
\right]
$$
$$
A\quad B\quad C\quad D
$$

components
Figure 2

This observation allows us to generalize such operations.

Definition 2.1: A set of machines and components whose deletion from the network yields disconnected networks is called a *bottleneck cell set*. If the bottleneck cell set contains only machines, then this set is a *bottleneck machine set*. A *bottleneck component set* is defined analogously.

As a consequence of Definition 2.1, one would like to find the smallest number of machines and/or components whose deletion disconnects the network. In addition, the designer may wish to control the number and size of the resulting disconnected networks. Thus, one would like to find the *minimal bottleneck cells* satisfying these system constraints.

Definition 2.2: A *minimal bottleneck cell set* is the smallest (in cardinality) bottleneck cell set whose deletion disconnects a machine-component representation into m disconnected subnetworks, each having at most k machines and components.

The importance of the minimal bottleneck cell set is that is *quantifies* how far an operation is from being disconnected. A small minimal bottleneck cell set implies that it is easy to disconnect the operation; a large set implies that the existing operation may be difficult to disconnect.

Finally, an important consideration in completing the analysis is to determine what is to be done with the minimal bottleneck cells themselves. One action that may be taken with bottleneck machines is to duplicate or buy the necessary bottleneck machines to disconnect the components produced by the bottleneck machine. For instance, in Figure 1, one bottleneck machine #5 can process components A and B while another machine #5 can process components C and D. This is shown in Figure 3.

machines

$$
\begin{array}{c}
1 \\
2 \\
5(1) \\
\\
3 \\
4 \\
5(2)
\end{array}
\left[
\begin{array}{cc:ccc}
1 & 1 & & & \\
1 & & & & \\
1 & 1 & & & \\
\hdashline
 & & 1 & 1 & \\
 & & & 1 & \\
 & & 1 & 1 &
\end{array}
\right]
$$
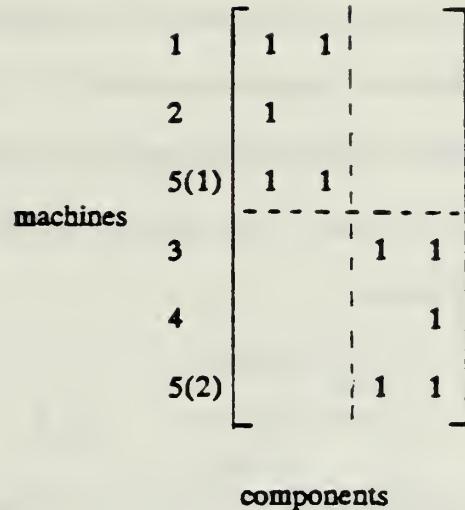
components

Figure 3

The advantages of more efficient scheduling and decentralized operations usually outweigh the cost of the extra machines in the long run.

For bottleneck components, a different approach is used. If a component requires too many operations on it to make it economically viable, then it may be more useful to "buy the parts instead of making them" as Burbidge (1977) suggests. The Japanese, in their automotive industry, have excelled in the use of subcontractors, using many more than their U.S. competitors while retaining quality and low cost characteristics. For example, in Figure 1, component E may be produced by a subcontractor.

3. Relating Bottleneck Cell Problems to Cutnode Problems

In this section, we describe an existing algorithm for finding minimal cutnodes of a graph subject to disconnecting the graph into m subgraphs having an upper bound k on the number of nodes in each subgraph. We show that this problem is *equivalent* to finding the minimal bottleneck cell set.

Consider the 0-1 matrix representation of the bipartite graph having p nodes connected to c nodes, where $a_{ij} = 1$, if node i is connected to node j. Given that the maximum number of rows and columns in each group is k, then the minimal cutnode problem is equivalent to permuting the rows and columns of A into a bordered block diagonal matrix $\overline{A}$ where the number of rows and columns in the border of $\overline{A}$ is *minimized.*

$$\overline{A} = \begin{bmatrix} \overline{A}_{11} & & & & & \overline{A}_{1,m+1} \\ & \overline{A}_{22} & & O & & \overline{A}_{2,m+1} \\ & & \ddots & & & \vdots \\ & O & & & & \\ & & & \overline{A}_{m,m} & \overline{A}_{m,m+1} \\ \overline{A}_{m+1,1} & \overline{A}_{m+1,2} & \cdots & \overline{A}_{m+1,m} & \overline{A}_{m+1,m+1} \end{bmatrix}$$
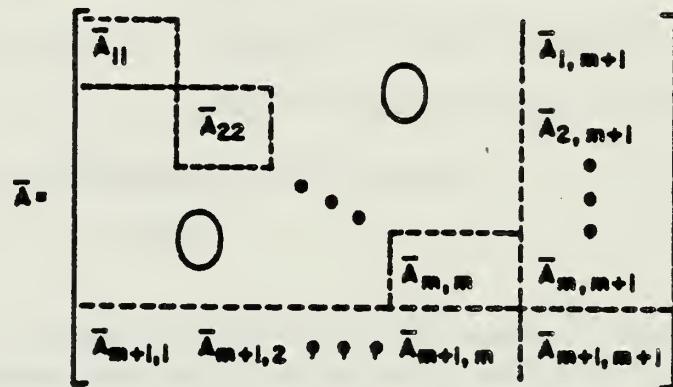
Figure 4

This formulation is equivalent to finding the minimal bottleneck cell set. Let p denote the number of processors or machines and c denote the number of components. Let A be a $p \times c$ matrix where

$$a_{ij} = \begin{cases} 1 & \text{if processor } i \text{ operates on component } j \\ 0 & \text{otherwise.} \end{cases}$$

Then, the minimal cutnodes of $\overline{A}$ equals the minimal bottleneck cell set.

Finding the minimum cutnodes in a graph is known to be an NP-complete problem. However, an efficient heuristic described in Lee, *et al.* (1982) has proven useful in approximating minimal cutnodes in several unrelated problem areas. This algorithm attempts to find the minimal cutnode set for m subgraphs having a constraint size k and an *initial* starting point in each subgraph. We present it here for completeness.

We begin with the following definition.

Definition 3.1 Let A be a subset of the nodes of a graph $G = (V, E)$. The *boundary node set* $\Gamma(A)$ is the set of nodes in $V \cap A^C$ which are connected to nodes in A. Given the node sets $\{A_1, A_2, \ldots, A_m\}$, the *uncommon boundary node* set $\Gamma''(A_i)$ is the subset of $\Gamma(A_i)$ such that if $v \in \Gamma(A_i)$, then $v \notin \Gamma(A_j)$, $j \neq i$.

Algorithm 1

Step 1: Given that the number of nodes N in a network and a node size constraint k for each subnetwork, calculate the minimal number of subnetworks given by

$$\bar{m} = \left[\frac{N-1}{K} \right] > 1$$

where $[(N-1)/K]$ indicates the closest integer no smaller than $(N-1)/K$.

Step 2: For m subnetworks, do Steps 3–6. Initially, $m = \bar{m}$.

Step 3: Choose m initial nodes, one for each subnetwork.

Step 4: Form a boundary node set for each subnetwork such that the removal of the set isolates the subnetwork from the rest of the whole network. Any node contained in more than two boundary node sets becomes a member of the cutset and it is removed from the list of boundary nodes.

Step 5: Select a node from each boundary node set and add it to the corresponding subnetwork such that the number of nodes of the resulting boundary node set decreases maximally or increases minimally.

Step 6: Repeat Steps 4–5 until every node is assigned to a subnetwork or cutset.

Step 7: Repeat Steps 3–6 to find the best solution. If a solution not violating the constraint is found, stop. Otherwise, increase m by 1 and go to Step 2.

Example 3.1

As an illustration of Algorithm 1, consider the following 5 node example given in Figure 5.
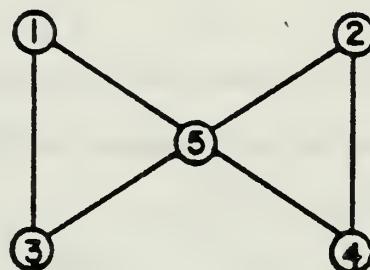
Figure 5

Suppose $K = 3$

Step 1: $\overline{m} = [(5 - 1)/3] = 2.$

Step 2: For two subnetworks, $m = 2.$

Step 3: Choose nodes 1 and 2 as respective starting nodes.

Step 4: Let

$$\Gamma(A) \triangleq \text{boundary node set of A}$$

$$\Gamma''(A) \triangleq \text{uncommon boundary node set of A}$$

$$\Gamma(\{1\}) = \{3, 5\}$$

$$\Gamma(\{2\}) = \{4, 5\}$$

Thus, node 5 is a cutnode since

$$\Gamma''(\{1\}) = \{3\}$$

$$\Gamma''(\{2\}) = \{4\}$$

Step 5: Add nodes 3 and 4 to {1} and {2} respectively.

$$\Gamma''(\{1, 3\}) = \Gamma''(\{2, 4\}) = \phi$$

Step 6: All vertices are assigned, node 5 is the only minimal cutnode.

Several properties of Algorithm 1 make it useful for grouping component-processor families. First, this heuristic is polynomially bounded. If v is the average number of edges attached to any node in the graph $G = (V, E)$, then it can be shown that the algorithm requires $O(v^m)$ storage and $O(v^m \log(v^m))$ running time. Since many component-processor networks are sparse, the average number of edges v and the number of groups m tend to be small.

Second, the initial starting points or seeds may be chosen from several practical considerations. Burbidge (1975, pp. 184-187) uses the "nuclear synthesis" method for grouping. The nucleus that he uses can be considered as the initial starting point in Algorithm 1. Alternatively, the physical characteristics of the components and/or processing features may be used to identify starting points. For example, circular and rectangular parts are more likely found in different groupings. Lathes and shaping machines exhibit the same behavior.

4. Finding Minimal Bottleneck Machines Only

Algorithm 1 allows the designer to find minimal bottleneck cell sets; however, in many instances the designer may wish to find *only* minimal bottleneck machine sets or bottleneck component sets. In the first case, he may wish to investigate the possibility of duplicating the smallest number of machines to disconnect his operation. In the latter case, he may wish to find the smallest number of components to subcontract outside and still disconnect the operation. Allowing the machines to be one set of nodes in a bipartite graph and the components to be the other set of nodes, the graph is depicted in Figure 6.

m-MACHINES   n-COMPONENTS
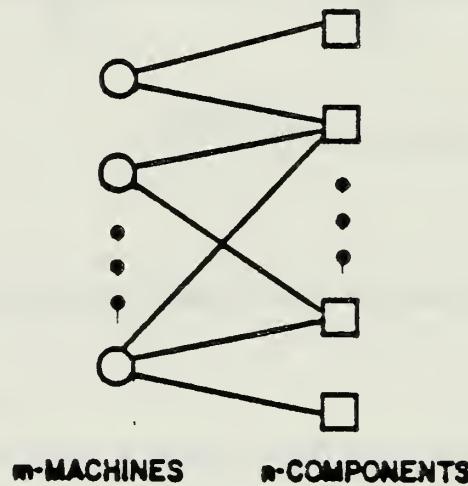
Figure 6

Unlike Algorithm 1 which does not distinguish between machine and component nodes, we wish to find the minimal cutnodes from *only one* set of nodes in a bipartite graph.

To accomplish this we *generalize* the notion of an edge between machines. Note that two machines m_1 and m_2 are linked only through components

m_1 m_2 MACHINES

COMPONENTS
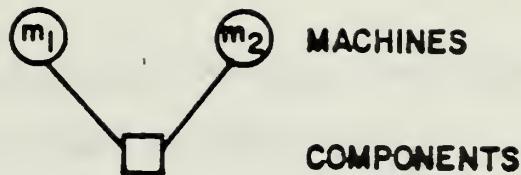
Figure 7

Thus we modify Algorithm 1 to choose cutnodes from one and only one set of nodes of the bipartite graph as follows. Algorithm 2 deals with the case of minimal bottleneck machine cell sets. A similar algorithm can be derived for bottleneck components.

Algorithm 2

Step 1: As in Algorithm 1, define

$$N_{man} \triangleq \text{number of machines}$$

$$K_{man} \triangleq \text{upper bound on machines in each group}$$

Choose corresponding number of initial groups $\overline{m}_{man}$, where

$$\overline{m}_{man} = \left[\frac{N_{man} - 1}{K_{man}} \right],$$

Step 2: Same as in Algorithm 1

Step 3: Choose $\overline{m}_{man}$ initial machines. Assure that selected machines are not linked through components. Add all the components attached to these machines.

11

Step 4: Same as in Algorithm 1. Note that only machine nodes are chosen at this stage.

Step 5: Select a machine node from each boundary node set and add it to the corresponding subnetwork such that the number of nodes of the resulting uncommon boundary node set is decreased maximally or increased minimally. If two boundary machine nodes are linked by a component, then only one of them may be added to the existing sets. Add the remaining component nodes attached to machine nodes that have just been added to m sets.

Note: This last operation along with the restriction that no two machine nodes have a common component are added to each network assure that cutnodes are *only* machine nodes.

Step 6: Same as in Algorithm 1.

Step 7: Same as in Algorithm 1.

Algorithm 2 is illustrated on the following example

Example 4.1. Consider the 5 component, 4 machine example given in Figure 8.

COMPONENTS   MACHINES

Figure 8

Let $K_{man} = 2$, $N_{man} = 4$
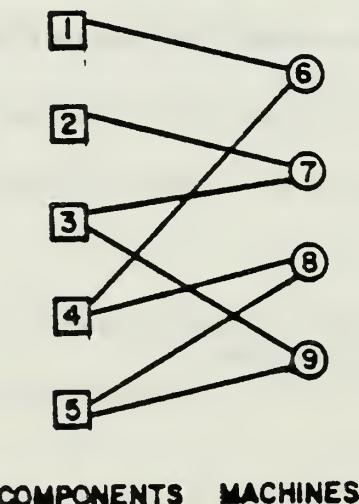
$$Step\ 1: \overline{m}_{man} = \left[\frac{4-1}{2}\right] = 2$$

Step 3: Choose machines 6 and 7 as initial machines. Note they are not linked by components. Attach components 1, 4 to machine 6 and components 2, 3 to machine 7.

Step 4: $\Gamma(\{6, 1, 4\}) = \{8\} = \Gamma''(\{6, 1, 4\})$

$$\Gamma(\{7, 2, 3\}) = \{9\} = \Gamma''(\{7, 2, 3\})$$

Step 5: Since machines 8 and 9 are *linked* by component 5, choose to add only one to the existing sets, say machine 8. Thus the existing sets are augmented to

$$\{6, 1, 4, 8, 5\},\ \{7, 2, 3\}$$

Step 6: The minimal bottleneck machine is machine 9.

Algorithm 2 has an additional important feature. Since the number of machines is small in real-life examples; i.e., in the order of 200 maximally, the polynomial bounded storage and running time allow even the APL test codes to run quickly. We show this in the next section.

5. Numerical Results

Algorithms 1 and 2 have been coded in APL and were run on an IBM 4341 machine at T.J. Watson Research Center. Algorithm 1 has also been coded in FORTRAN (Lee, *et al.*, 1979). The two codes have been tested on many medium to large-sized problems. We report results of these two algorithms on two test cases found in the existing literature.

Example 5.1

The first example is a 16 machine-43 component example in King and Nakornchai, (1982, pp. 128-129). This GT problem is shown in Figure 9. King's ROC algorithm decomposes this problem into 4 groups. The bottleneck cells set contains the machines 6 (which is duplicated two times) and 8 (which is duplicated three times). An exceptional element exists in the final decomposition depicted in Figure 10.

components
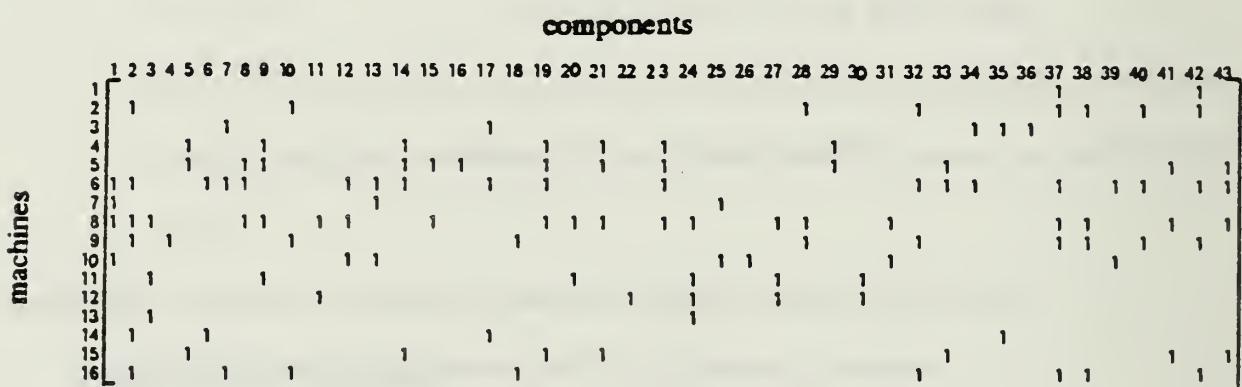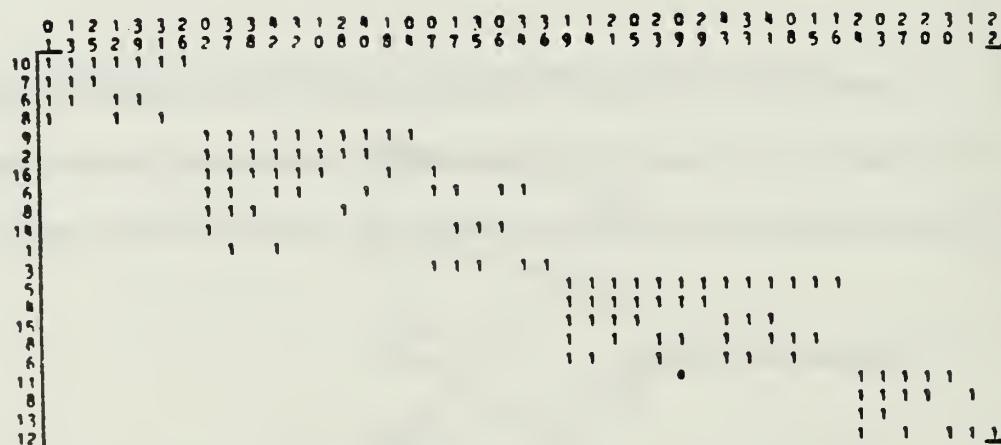
Figure 9

Figure 10

On applying Algorithms 1 and 2 with the starting machine nodes 1, 4 and 7, $m = 3$, and $K = 30$, the minimal bottleneck cell set and minimal bottleneck machines set contain machines 6 and 8 also. However, unlike King's results, no exceptional elements have to be dealt with after duplication. Each machine is duplicated *two* times. See Figure 11. The running time was 1.01 seconds CPU time.

components
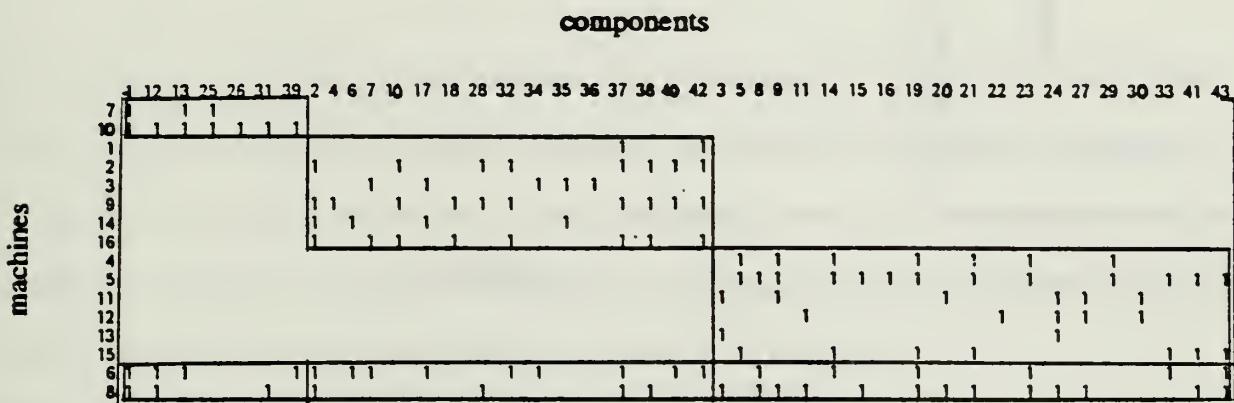
Figure 11

Example 5.2

The following real-life example considered by Burbidge (1973) illustrates the order of magnitude improvement that is often possible with these algorithms. Burbidge consider the following data from Black and Decker Ltd. (see Figure 12). Machines 3, 4, 8, 10, 12 and 24 were not considered since they did not process components. On applying Algorithm 2 with starting machine nodes 7 and 17, $m_{man} = 2$, and $K_{man} = 26$, the minimal bottleneck machines were found to be 1, 31, 32, 33, 34. Thus, if these five machines are duplicated, perfect decomposition results. This is shown in Figure 13. The running time was 2.65 seconds CPU time. King and Nakornchai (1982) need to duplicate *eighteen* machines with the ROC approach. They obtain six exceptional elements also.
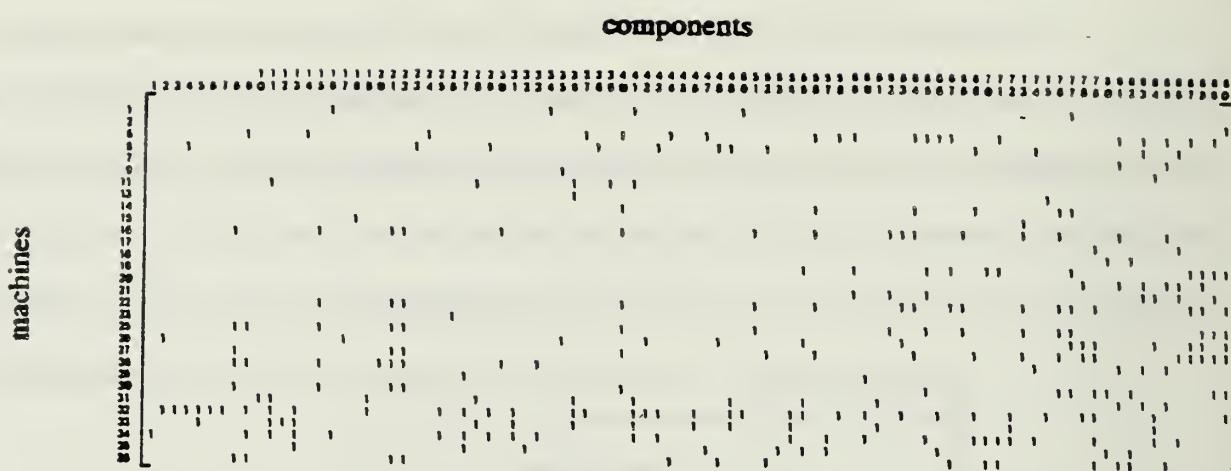
components
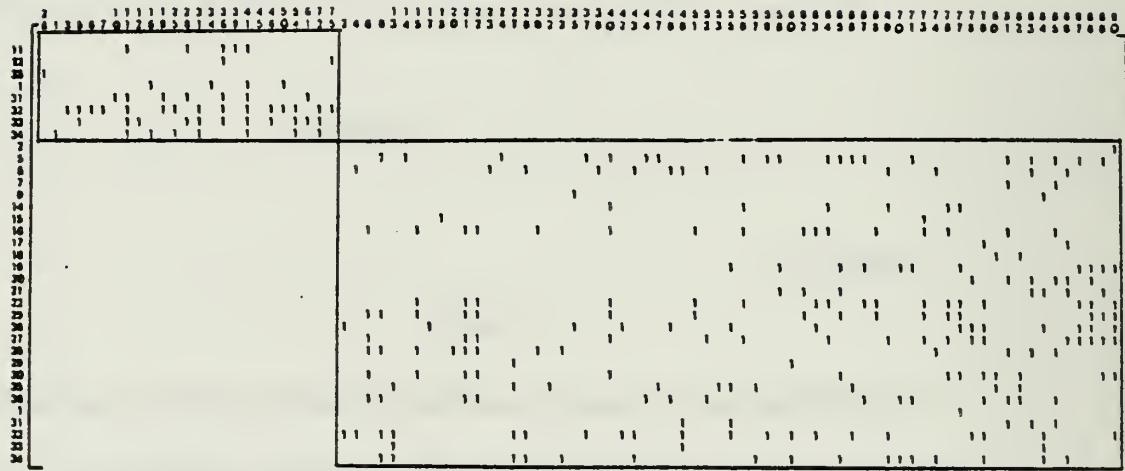
Figure 12

Figure 13

6. Conclusions

In every manufacturing facility, two commonly found characteristics are component subcontracting and machine duplication. The former is usually used as a strategy to reduce capacity requirements while the latter is used to increase capacity available.

In this paper, we have outlined the use of these strategies to streamline the manufacturing system. The production system, comprising of components and processors, are grouped to form component-

processor families so as to reduce interaction between families. The two strategies of subcontracting and duplication can be used to perfectly decompose the system into non-overlapping families. This decomposition allows the management to contemplate the use of cellular manufacturing technologies or even the implementation of flexible manufacturing systems on each of the component-processor families.

The choice of the strategy, either subcontracting or duplication or both, is left to the designer. In most cases, the economics of machine duplication and the strategic/economic circumstances of subcontracting will dictate the best way to group component-processors. The algorithms developed in this paper will serve as a decision aid to generate possible groupings and the minimal bottleneck cells; i.e., components to be subcontracted or processors to be duplicated.

The algorithms suggested are very efficient and perform a better partition on a 16 machine-43 component problem and on a 36 machine-90 component problem than the clustering algorithms proposed by King (1980), King and Nakornchai (1982) and Burbidge (1973). An order of magnitude improvement can be obtained by translating these APL codes into higher-level language codes. This is the next phase of the research. Based on this limited testing, the storage and running time considerations, the proposed algorithms may serve as important decision aids for a designer of cellular manufacturing systems or a flexible manufacturing system.

References

Barnes, E.R. (1982), An algorithm for partitioning the nodes of a graph, *SIAM J. of Alg. and Discrete Methods*, 3, 541-550.

Basale, P.C. and Tripathi, D.K. (1981), Group technology in CRS through PFA, *J. of Inst. of Engineers (India) - M.E.*, 61, 204-206.

Burbidge, J.L. (1973), Production flow analysis on a computer, *Third Annual Conference of the Institution of Production Engineers*.

Burbidge, J.L. (1975), *The Introduction of Group Technology*, Halsted Press, John Wiley and Sons, New York.

Burbidge, J.L. (1977), A manual method of production flow analysis, *Prod. Engineer*, 56, 34.

Chakravarthi, A.K. and Shtub, A. (1984), Selecting parts and loading flexible manufacturing systems, *Proc. of First ORSA/TIMS Special Interest Conference on FMS*, Ann Arbor, Michigan.

Harary, F. (1959/60), A graph theoretic method for the complete reduction of a matrix with a view toward finding its eigenvalues, *J. of Math. and Physics*, 38, 104-111.

King, J.R. (1979), Machine - component group formation in group technology, *Fifth International Conference on Production Research*, 40-44.

King, J.R. (1980), Machine - component grouping in production flow analysis: an approach using a rank order clustering algorithm, *Int. J. of Prod. Res.*, 18, 213.

King, J.R. and Nakornchai, V. (1982), Machine - component group formation in group technology: review and extension, *Int. J. of Prod. Res.*, 20, 217.

Kumar, K.R. and Vannelli, A. (1984), Efficient algorithms for grouping component-processor families, IBM Research Report No. RC 10636 (#47695), Thomas J. Watson Research Center.

Kusiak, A. (1984), The part families problem in flexible manufacturing systems, Working Paper #06/84, Technical University of Nova Scotia, Dept. of Industrial Engineering.

Lee, J.G., Vogt, W.G., and Mickle, M.H. (1979), Optimal decomposition of large-scale networks, *IEEE Trans. on SMC*, SMC-9, 7, 369-375.

Lee, J.G., Vogt, W.G., and Mickle, M.H. (1982), Calculation of shortest paths by optimal decomposition, IEEE Trans on SMC, SMC-12, 3, 410-415.

Ogbuorbiri, E.C., Tinney, W.F., and Walker, J.W. (1970), Sparsity - directed decomposition for Gaussian elimination on matrices, *IEEE Trans. on PAS*, PAS-81, 141-150.

Steward, D.V. (1962), On an approach to techniques for the analysis of the structure of large systems of equations, *SIAM Review*, 4, 321-342.

Steward, D.V. (1965), Partitioning and tearing systems of equations, *J. Numer. Anal.*, 2, pp. 345-365.

Vannelli, A. (1984), Approximating a class of graph decomposition problems by linear transportation problems, *TIMS/ORSA Meeting San Francisco*, May (1984); IBM Research Report No. RC 10584 (#47380).

Zaborsky, J., Whang, K.W., Huang, G.M., Chiang, L.J., and Lin, S.Y. (1982), A clustered dynamic model for a class of linear autonomous systems using simple enumerative sorting, *IEEE Trans. on CAS*, CAS-29, 747-758.